SECURITY CLASSIFICATION OF THIS PAGE

4

REPORT DOCUMENTATION PAGE

AD-A196 102

DTIC FILE COPY

1b RESTRICTIVE MARKINGS			
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE		3. DISTRIBUTION / AVAILABILITY OF REPORT unlimited	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)		5. MONITORING ORGANIZATION REPORT NUMBER(S)	
6a. NAME OF PERFORMING ORGANIZATION The Regents of the University of California	6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION SPAWAR	
6c. ADDRESS (City, State, and ZIP Code) Berkeley, California 94720		7b. ADDRESS (City, State, and ZIP Code) Space and Naval Warfare Systems Command Washington, DC 20363-5100	
8a. NAME OF FUNDING / SPONSORING ORGANIZATION DARPA	8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER	
8c. ADDRESS (City, State, and ZIP Code) 1400 Wilson Blvd. Arlington, VA 22209		10. SOURCE OF FUNDING NUMBERS	

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)

* PARALLEL ALGORITHMS FOR GROEBNER-BASIS REDUCTION

12. PERSONAL AUTHOR(S)
* Carl Ponder

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
technical	FROM ________ TO ________	* September 25, 1987	* 17

16 SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Enclosed in paper.

DTIC
S ELECTE D
JUL 2 6 1988
E

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code) / 22c. OFFICE SYMBOL

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE

Productivity Engineering in the UNIX† Environment

Parallel Algorithms for Groebner-Basis Reduction

Technical Report

S. L. Graham
Principal Investigator

(415) 642-2059

Contract No. N00039-84-C-0089

August 7, 1984 - August 6, 1987

Arpa Order No. 4871

†UNIX is a trademark of AT&T Bell Laboratories

Parallel Algorithms for Gröbner-Basis Reduction

Carl Ponder
Computer Science Division
Electrical Engineering and Computer Sciences
University of California, Berkeley
Berkeley, CA. 94720

September 25, 1987

Abstract [illegible]

We present a variety of ways to parallelize Gröbner-basis reduction, ranging from incorrect to ineffectual. ~~We demonstrate~~ the superiority of the method used by Zacharias ~~[1]~~, which is not readily parallelizable. We discuss the efficiency issues of generating reduced Gröbner-bases. [illegible]

Categories and Subject Descriptors: G.4 [**Mathematics of Computing**]: Mathematical Software; I.1.1 [**Algebraic Manipulation**]: Expressions and Their Representation; I.1.2 [**Algebraic Manipulation**]: Algorithms; J.2 [**Computer Applications**]: Physical Sciences and Engineering.

General Terms: Algorithms, design.

Additional Key Words and Phrases: Computer algebra, Gröbner bases, parallel computation, polynomial ideals, simplification.

1 Introduction

Gröbner-basis reduction would be a powerful tool for solving problems in connection with systems of multivariate polynomials [2] [3] [4] if it weren't so costly. Two important uses are for solving systems of nonlinear equations with arbitrarily-many variables and arbitrary degree, and for simplification of polynomial expressions subject to polynomial equality side-relations. A number of important problems can be reduced to Gröbner-basis computations, although this is not necessarily an efficient reduction [2]. It can be used in conjunction with other (more specialized) techniques as part of a general equation solver [5].

Informally, given a field K and set of variables $\{x_1, ..., x_n\}$, and an ordering relation between the products of the variables, the Gröbner-basis reduction maps each (finite) set of polynomials over the ring $K[x_1, ..., x_n]$ to a canonical (finite) set of polynomials from the ring, which generates the same ideal within the ring.

Buchberger presents a simple algorithm for producing the reduced basis. As we shall see, the algorithm cost can be quite large. The (at least) doubly-exponential time complexity [6] holds up in practice. As a result, any opportunity to speed up this potentially highly-useful process is welcome.

We programmed some plausible ways to run the algorithm in parallel, including a method suggested by Watt [7]. Experiments suggest that an efficient serial method such as used in the Macsyma package [1] is difficult to beat, in practice, with any modest amount of parallelism. These disappointing results are explained in this paper.

2 Gröbner-Basis Reduction

Formally, we describe the Gröbner-basis reduction process as follows: Given a field K, let $R = K[x_1, ..., x_n]$ be the ring of polynomials in n indeterminates or variables over K. Often we take K as the rationals. A monomial term has the form $\alpha = kx_1^{e_1}...x_n^{e_n}, k \in K$. The degree of a monomial is the sum of its exponents $e_1+...+e_n$. The degree of a polynomial is the maximum degree over all its monomials. Let "<" be a total order on the monomial terms that is preserved under multiplication by a variable. Each polynomial can then be thought of as a *lead term* (the dominant monomial) plus the remaining monomials. Two possible orders $<$ are *lexicographic* and *total degree*.

In lexicographic ordering, we order the variables $x_1...x_n$ so $x_{\pi_1} < x_{\pi_2} < ... < x_{\pi_n}$ for some permutation π. For two monomials $\alpha \neq \beta$, we say $\alpha < \beta$ if α is of lower degree than β in the most dominant variable in which the exponents of α and β differ. In total-degree ordering, $\alpha < \beta$ if the (cumulative) degree of α is less than that of β; ties are broken lexicographically.

The ordering generalizes to polynomials by internally placing the monomials terms in decreasing order. A polynomial α is $<$ a polynomial β if the leading term of α is $<$ the leading term of β. Ties are broken by comparing the next terms in sequence.

The Gröbner-basis reduction consists of two steps: *generation of S-polynomials*, and *reduction to normal form*. Given polynomials p and q, the S-polynomial of p and q is $ap - bq$ where a and b are chosen so that ap and bq have the same leading monomial, which is the least common multiple of the leading monomials of p and q. Thus we make the leading terms cancel.

A polynomial p can be *reduced* (with respect to a basis B) to a polynomial q if $q < p$ and there is a monomial c and a polynomial $r \in B$ such that $p - cr = q$. A polynomial p is in *normal form* if it cannot be reduced with respect to B.

The Gröbner-basis reduction algorithm begins with an initial basis B, and proceeds by generating pairwise S-polynomials of the elements of B, entering them into B, and reducing each element of B with respect to the remaining *elements of B*. It completes when all the basis elements in B are reduced with respect to each other and the S-polynomial of each pair in B reduces to zero.

The (Buchberger) algorithm is shown here in figure 1. In section 4 we will show some ways of parallelizing this algorithm. Although there may be better approaches, we have not found substantially better algorithms to achieve the same result. We discuss the complexity of the Grõbner-basis reduction problem in the next section.

```
Input:   A set of polynomials $F$ over $R$, an ordering $<$
Output:  $G$, a set of polynomials representing the reduced basis of the
         ideal generated by $F$.

         $G:=F$;
         $B:= \{(f_1, f_2) | f_1, f_2 \in G, f_1 \neq f_2\}$
         While $B \neq \phi$ do
              let $(f_1, f_2)$ := a pair in $B$;
              $B := B - \{(f_1, f_2)\}$;
              $h$ := SPolynomial($f_1$,$f_2$,$<$);
              $h'$ := NormalForm($G$,$h$);
              if $h' \neq 0$ then
                 $B := B \cup \{(g, h') | g \in G\}$;
                 $G := G \cup \{h'\}$;
```

Figure 1 – Buchberger's algorithm for performing Grõbner-basis reduction

3 Complexity

Tight upper and lower bounds on the complexity of Grõbner-basis reduction are not known. The algorithm in figure 1 may not be the most efficient. Bounds on the size of the reduced basis are presented in [8] and [6]. Given a basis B, the *cardinality* m is the number of polynomials in B. The *degree* d is the maximum degree over all the polynomials in B. The *alphabet* is the set of variables, and has size n. The *dimension* s of the ideal is the minimum cardinality of any generator for it. Let B' be the reduced Grõbner-basis of B. An upper bound on the degree of B' [8] is

$$O(((n+1)(d+1)+1)^{(n+1)2^{s+1}})$$

One worst-case lower bound on the degree of B' [8] is

$$d^{n-1}(d^2+1)$$

and another worst-case lower bound on both the degree and cardinality of B' [6] is

$$2^{2^m}$$

These bounds consider only the size of the basis produced. The size provides a strict lower bound on the time required to produce the reduced Gröbner-basis, but there is no reason to believe any algorithm can achieve any such efficiency.

In practice, finding the reduced Gröbner-basis of a small set of polynomials can take a huge amount of time. For example, in test case #6 we had three polynomials in five variables, where the maximum total degree of any monomial is two. Computing the reduced basis took over 6 minutes on a VAX 8600, *not* including garbage-collection time.

Predicting how long the reduction will take is rather difficult. Between test cases #5 and #6, the difference in computing time was a factor of over fourteen, caused by the introduction of a single variable:

$$\left\{ \begin{array}{l} x^2 + y^2 + z^2 - t \\ y^2 - y + z - x^2 - x - 1 \\ x^2y^2 - 1 \end{array} \right\}$$

$$\left\{ \begin{array}{l} x^2 + y^2 + z^2 - t \\ y^2 - y + z - x^2 - x - s \\ x^2y^2 - 1 \end{array} \right\}$$

(The basis is taken over $K(t)[x, y, z]$ in the first example and $K(s, t)[x, y, z]$ in the second). The cost of the reduction is highly sensitive to the form of the initial (input) basis. Two sequences of reductions to the same final basis may take wildly different amounts of time if they produce two distinct intermediate bases at any stage.

In most cases the length of time required to reach the final basis will depend upon the order in which the S-polynomials are generated and reduced [2]. The ordering of the terms of the polynomial will direct the order in which reductions are performed. The choice of total degree or (which) lexicographic order will make a significant difference in the time taken (see section 7). For example, in a set of polynomials with a dependent variable, taking that variable as dominant in a lexicographic order will cause it to be eliminated entirely, producing a simpler basis. This should take longer in any other ordering, since that variable will only be eliminated as a secondary priority.

Buchberger's algorithm amounts to a search for the reduced basis by testing at each stage whether new S-polynomials reduce to 0 or not. Each step of producing and entering a reduced S-polynomial causes the intermediate basis to converge toward the final basis, although the process is very slow. The Buchberger algorithm doesn't specify which choice of two polynomials are to be operated on. We will see later how two algorithms that order the S-polynomial operations and reductions differently will differ in the number of such operations they perform. Under different test cases each can generate fewer S-polynomials or require fewer reductions than the other. A fruitful direction may be to find some good heuristics that can be used to steer the derivation in a more efficient direction. In some ways it is reminiscent of the Simplex algorithm, which repeatedly selects a pivot equation and runs until an optimal state is found. The discovery of efficient alternatives to Simplex coupled with our poor understanding of the complexity of Gröbner-basis reduction leaves the nagging suspicion that there may be methods superior to Buchberger's, at least for cases of practical interest. Intuition can sometimes be helpful in suggesting subdivisions of problems which save enormous amounts of computation.

4 Parallel Variations of Buchberger's algorithm

We have seen that the Gröbner-basis reduction in general has been shown to be inherently hard. The algorithm stated by Buchberger is quite simple. The structure of the inner loop suggests that it might be made to run in parallel; further, regarding the algorithm as a search allows us to suppose much of the work is divided between unrelated activities, such as generating pairwise S-polynomials and reducing various basis elements against each other. For example, confirming that a basis is in reduced Gröbner form can be parallelized quite effectively. The S-polynomials can all be generated and reduced against the basis elements simultaneously, while the basis elements are reduced against each other. The basis is in reduced Gröbner form if and only if all the S-polynomials reduce to zero and no basis element is reducible. This leads us to propose these three ways to parallelize Gröbner-basis reduction:

[a] Compute the S-polynomials in parallel, and reduce the basis in serial.

[b] Generate the S-polynomials one at a time, and use the result to reduce each basis element simultaneously.

[c] Divide the process into alternating stages of S-polynomial generation and reduction, and use parallelism in each stage.

We can add two other unrelated ways of using parallelism:

[d] Repeatedly find the reduced Gröbner-bases of different subsets of the basis and merge them, until the basis converges.

[e] Simultaneously reduce the basis under different orderings.

On first examination method [c] appears to be the most promising. In fact, this was proposed by Watt [7] in his Ph.D. dissertation. Methods [a] and [b] look quite a bit weaker. We will see the faults of method [c] and discuss [a] and [b] as alternatives. In section 6 we will show some empirical results. [e] is deferred to section 7.

Consider [d]. An important fact about the Buchberger algorithm is that key polynomials are formed which cause the basis elements to reduce drastically. For example, once a variable is isolated so it appears only in the head term of a polynomial it will cause all other instances of the variable to be eliminated. This elimination property suggests that the basis should be kept as far reduced as possible so reductions can take effect as soon as possible. Splitting the basis will deprive some polynomials of the chance to get reduced until a later stage. It is likely that such a technique would take much longer to converge because of this. Alternately, the separate bases can be merged and the result fully reduced, but this converts the algorithm to one more like [a].

```
$G' := \phi$;
$G := F$;
while $G \neq G'$ do
        $G' := G$;
        $B := \{(f_1, f_2) | f_1, f_2 \in G, f_1 \neq f_2\}$
        $H := G$;
        forall $(f_1, f_2) \in B$ do
                $h :=$ SPolynomial($f_1$,$f_2$);
                $h' :=$ NormalForm($G$,$h$);
                if $h' \neq 0$ then $H := H \cup \{h'\}$;
        $G := \phi$;
        forall $h \in H$ do
                $h' :=$ NormalForm($H - \{h\}$,$h$);
                if $h' \neq 0$ then $G := G \cup \{h'\}$;
return $G$;
```

Figure 2 – Watt's parallel algorithm

The method [c] proposed by Watt was expressed as the algorithm shown in figure 2. Unfortunately it doesn't work. The logic seems fairly straightforward: simultaneously generate each S-polynomial and reduce it with respect to the previous basis. Add these new elements to the basis, and fully reduce the basis by simultaneously reducing each basis element with respect to the other basis elements until nothing reduces any further. The problem is that reducing basis elements with respect to each other produces the wrong results. A crude example is to start with the same element twice:

$$\left\{ \begin{array}{c} x+y \\ x+y \end{array} \right\}$$

The reduced Gröbner-basis is the set $\{x+y\}$. But reducing the two polynomials with respect to each other causes both to vanish, giving an empty basis. This is wrong. The essential detail is that each polynomial must be deleted as it is reduced; Buchberger's algorithm enters the reduced version back into the basis immediately, to help reduce the remaining elements. Alternatively, the newly reduced element may be held outside the basis to be added later. Such an approach resembles suggestion [d], and would probably delay convergence.

Suggestions [a] and [b] are attempts to salvage the ideas from [c]. Tests of these ideas are presented in section 6. It is essential that they keep the basis fully reduced after the S-polynomial(s) stage; otherwise convergence is delayed. It was found that on most of the test cases either space was exhausted or the computation took too much time if the basis was not kept reduced.

The insistence on keeping the basis fully reduced limits our ability to parallelize. We cannot simultaneously reduce all the elements with respect to each other. Thus we have some implicit serialization going on; we need to treat each basis element with respect to a set that is changed by our treatment of the previous basis element. Let us look at the other methods in detail.

Suggestion [a] computes the S-polynomials in parallel:

```
$G' := \phi$;
Repeat until $G = G'$
        $G' := G$;
        Simultaneously generate all S-polynomials from basis $G$,
                        and reduce the result w.r.t. $G$;
        Insert each S-polynomial into $G$;
        Fully reduce $G$;
Return $G$;
```

There are two problems with this. First, each new S-polynomial might reduce the basis enough that the other S-polynomial operations are redundant. This will again delay the convergence. Second, reducing the basis in serial will add a serial step to the process, limiting the effect of parallelization. As we shall see, the performance of this algorithm was generally poor.

Suggestion [b] computes each S-polynomial one at a time, assuming that the basis is changing significantly each time a S-polynomial is produced and used to reduce the other basis elements. It is reduced against the basis, and used to reduce each of the remaining basis elements in parallel. This may produce new reduced elements, which are then put through the same process:

```
Repeat until $G$ no longer can change
        Choose two polynomials $f$ & $g$ from $G$;
        Take their S-polynomial $h$;
        Reduce $h$ w.r.t. $G$;
        Simultaneously reduce each element of $G$ w.r.t. $h$;
        Insert $h$ into $G$;
        Reduce $G$ w.r.t. itself;
Return $G$
```

The problem with this is that in the later stages of the process, most S-polynomials will reduce to zero, so the parallel step will not be used. In the earlier stages of the process, most S-polynomials will have degrees too large to be used for reduction, even after they are reduced. The only time the parallelism pays off is when the "magic" S-polynomials appear which cause the basis to begin collapsing. As we will see later on, the performance is again quite unimpressive.

5 Comparison with the Zacharias Implementation

Gail Zacharias [1] wrote a Macsyma package for performing Gröbner-basis reduction. It employs some interesting tricks which give it better performance than the serial or parallel versions of the algorithms presented previously. These "hacks" interfere with parallelism, so cannot be used in the other algorithms.

The basis is not kept in fully reduced form. Newly formed S-polynomials are fully reduced by the old basis, but the old basis is not immediately reduced by the result. This is done as part of the later stages. If one element's head is reducible by the other, the S-polynomial will be the reduction of the first with respect to the second. Thus full reduction is accomplished over several iterations. The Zacharias algorithm eliminates the step where the new S-polynomial is used to reduce the basis elements. This is the pivotal step in our second parallel algorithm, which turns out to be a liability.

When the S-polynomial operation reduces one of the elements, we do this destructively so subsequent S-polynomial operations simply pick up the reduced version. This is incompatible with our algorithm for producing all S-polynomials in one parallel sweep, since all these destructive operations would interfere with each other. The basis must be held in a reduced form between S-polynomial sweeps in our first parallel algorithm. When we do not do so, the process often does not converge within a reasonable amount of time. In some sense we are reducing the basis and generating S-polynomials "at different rates". If we do not do enough reduction, the S-polynomial operations will expand the basis indefinitely. If we do all the S-polynomial operations in parallel, we slow the rate of reduction since reductions are not performed immediately. We must make up for this by taking extra effort to reduce the basis between S-polynomial sweeps.

The S-polynomial of two basis elements x and y is not formed if there is a third basis element z such that the leading term of z is $<$ the LCM of the leading terms of $x\&y$, and the S-polynomials of x, z and y, z have been formed. This criterion is is used in Czapor and Geddes [9], along with the restriction that the S-polynomial of $x\&y$ is not formed if the two have no common factor. The Zacharias package only partially implements the second restriction. Czapor and Geddes report a fairly consistent factor of two speedup due to these restrictions. If anything, these restrictions should reduce the amount of parallelism by reducing the number of S-polynomials generated at each step.

6 Empirical Results

We show the performance aspects of three algorithms. These are referred to as *Zacharias*, *Parallel Reduction*, and *Parallel S-polys*, respectively. The first is the program written by Gail Zacharias, using the techniques described in the previous section. The second is method [b] which uses each new S-polynomial to reduce the remaining basis elements in parallel. The third is method [a] which produces all S-polynomials in simultaneous sweeps. Both the Parallel Reduction and Parallel S-Poly methods keep the basis fully reduced.

The programs were tested by coding them in Franz Lisp and loading them into Zacharias' Macsyma package for Gröbner-basis reduction. These were loaded into Vaxima version 2.11 running under Franz Lisp Opus 42 and Unix 4.3 BSD, on the Vax 8600 "Vangogh" at UC Berkeley. The test cases used are labeled 1-12, and are shown in the appendix. Strict lexicographic ordering was used, following alphabetical order.

Table 1 shows the running time of the three algorithms, in seconds. The serial time is shown for all three, which is the total time (minus garbage-collection time) to reduce each basis. The "parallel" time is computed by timing each iteration of the parallelizable loops, and only counting the slowest iteration. The Zacharias algorithm generally ran much faster than the other two, as much as 36 times faster in case #9. It only ran slower in case #4. Case #12 exhausted memory space in the parallel algorithms.

The "parallel" algorithms exhibited very little parallelism. The speedups in table 1 are generally insignificant. Table 2 shows the maximum and average parallelism for their executions. The "maximum" parallelism is the number of iterations performed by the parallel loop in a given activation. In the Parallel Reduction algorithm a low parallelism meant that there were few polynomials in the basis with degree higher than the new S-polynomial. In the Parallel S-Polys algorithm a low parallelism meant that the basis consisted of few elements at any given time. The "average" parallelism is the serial time divided by the parallel time. It tended to be very small.

Table 3 shows the number of S-polynomials generated and reductions performed by the algorithms, as well as the total number of terms in the result. The speed does not correlate well with either the number of S-polynomials or the number of reductions; the amount of time each of these operations takes, however, depends on their sizes. The total number of terms in the final basis is a strong hint of the time taken, although there may be very large intermediate polynomials which collapse down by the end.

The number of S-polynomial operations gives an idea about the size of the basis. In no case did it get very large. This is a disturbing observation about the problems; the algorithm can take a long time even though the basis is always small and remarkably few S-polynomial operations occur. Since these "sparse" problems are so expensive in practice, the "bad" cases will probably be even worse. As the number of polynomials, their degrees, and the number of variables increases, it is likely that the number of terms in the intermediate polynomials will grow much more quickly.

There seems to be very little easy parallelism in the light of our studies. Most of the time is probably spent scanning the terms of a polynomial to find monomials that reduce. This operation changes the lower-order monomials, so subsequent reductions may be possible. This cannot be done to each monomial simultaneously since new monomials get introduced with each reduction. The process will still be serialized by the introduction of new, possibly reducible, monomials with each step.

An interesting fact to note is that the relative number of S-polynomial and reduction operations changes from case to case. As we stressed before, simultaneously taking all S-polynomials of the basis will probably slow convergence since we lose the benefit of reducing the basis immediately each time an S-polynomial is generated. But cases #4 and #10 violate this intuition; fewer S-polynomials get generated under the Parallel S-polys algorithm than under the other two. Furthermore, in case #4 it even uses fewer reduction operations. It goes to show that choosing an optimal order for forming S-polynomials and reducing is not an easy thing to do.

Table 1 - Comparative running times (in seconds)

Test case	Zacharias time	Parallel Reduction serial time	Parallel Reduction parallel time	Parallel S-Polys serial time	Parallel S-Polys parallel time
1	0.316	0.600	0.600	0.416	0.316
2	0.416	1.083	1.066	1.250	0.933
3	0.216	0.383	0.366	0.383	0.333
4	1.716	1.316	1.233	0.833	0.766
5	27.266	125.533	119.616	171.750	108.250
6	381.950	1102.650	1040.467	1376.117	1001.434
7	11.350	34.850	32.817	36.867	36.867
8	31.416	569.983	569.033	541.300	523.050
9	9.416	416.450	277.817	342.516	337.033
10	69.417	727.817	714.700	147.550	133.717
11	173.234	641.867	641.334	547.917	386.267
12	3473.466	*	*	*	*

Garbage-collection time has been deducted from the execution.
In case 12 the two parallel schemes exhausted memory space before finishing.

Table 2 - Estimated parallelism in Parallel S-poly and Reduction algorithms

Test case	Parallel Reduction maximum	Parallel Reduction average	Parallel S-Polys maximum	Parallel S-Polys average
1	3	1.00	6	1.32
2	4	1.02	3	1.34
3	4	1.05	3	1.15
4	5	1.07	6	1.09
5	4	1.05	3	1.59
6	4	1.06	3	1.37
7	3	1.06	1	1.00
8	3	1.00	1	1.03
9	4	1.50	3	1.02
10	5	1.02	3	1.10
11	6	1.00	15	1.42

Table 3 – Other execution statistics							
Test	Zacharias		Parallel Reduction		Parallel S-Polys		# of terms
case	S-Polys	Reductions	S-Polys	Reductions	S-Polys	Reductions	in result
1	3	10	10	35	9	33	12
2	1	24	4	65	6	110	23
3	1	25	5	50	6	54	12
4	14	225	12	92	9	82	15
5	2	188	4	187	6	362	210
6	2	189	4	189	6	365	918
7	2	108	1	43	1	43	7
8	1	14	2	23	2	23	130
9	10	330	5	215	6	286	26
10	9	100	8	90	6	92	131
11	14	159	25	251	28	303	110
12	10	250	?	?	?	?	*

* Case #12 generated too many terms to deal with, well in the thousands.

7 Reducing Under Alternative Orderings

An alternative way to introduce parallelism is to simultaneously reduce a basis under different orderings, using whichever result comes first. This approach is referred to as *collusion* in [7]. Table 4 compares the time to reduce for several of the test cases (the table is incomplete because some cases took an excessive amount of time).

Cases 2, 3, 5, 7, and 11 were very sensitive to the ordering used. Case 11 is interesting in that no "intuitive" pairwise ordering between the variables seems to explain the results. Cases 5 and 7 appear to be strongly sensitive to the dominant variable. In the remaining cases 1, 4, 8 and 9 the ordering was of little consequence. There may be useful heuristics for selecting a reasonably efficient ordering given the form of the expressions in the initial basis.

Using one processor for reduction under each ordering would generate a solution in as much time as the fastest of the orderings. This would yield a reasonable speedup for cases 2, 3, 5, 7, and 11 compared to using the wrong ordering. If the number of processors is considered, cases 2 and 3 would gain at most a speedup of 4 for 6 processors, which is hardly worth the effort. Comparing the minimum time to the average time in cases 5, 7, and 11 shows that the "average" order would still take at least as long as the fastest time times the number of processes.

This technique is only useful if *any* reduced Gröbner basis is sufficient. For example, if we want to know if two sets of polynomials generate the same ideal,

Table 4 – Times (in Seconds) to Reduce Under Different Lexicographic Orderings

Test	Time	(Max Time)/(Min Time)	Ordering
1	0.316	1.26	$x > y > z > t > s$
	0.233		$x > z > y > t > s$
	0.300		$y > x > z > t > s$
	0.250		$y > z > x > t > s$
	0.250		$z > x > y > t > s$
	0.250		$z > y > x > t > s$
2	0.416	4.09	$x > y > z$
	0.416		$x > z > y$
	1.700		$y > x > z$
	0.583		$y > z > x$
	0.849		$z > x > y$
	0.499		$z > y > x$
3	0.216	4.86	$x > y > z$
	0.182		$x > z > y$
	0.550		$y > x > z$
	0.866		$y > z > x$
	0.482		$z > x > y$
	1.049		$z > y >$ [illegible]
4	1.716	1.02	$x > y$
	1.749		$y > x$
5	27.266	16.00	$x > y > z$
	29.033		$x > z > y$
	27.900		$y > x > z$
	27.399		$y > z > x$
	436.250		$z > x > y$
	424.983		$z > y > x$
7	11.350	8.41	$x > y$
	1.349		$y > x$
8	31.416	1.05	$x > y$
	33.049		$y > x$
9	9.416	1.13	$x > y > z$
	9.817		$x > z > y$
	10.450		$y > x > z$
	10.617		$y > z > x$
	10.133		$z > x > y$
	10.183		$z > y > x$
11	173.234	66.22	$x > y > z$
	18.450		$x > z > y$
	72.434		$y > x > z$
	7.967		$y > z > x$
	2.616		$z > x > y$
	25.434		$z > y > x$

we reduce both and check to see if the reduced forms are identical. If we reduced each under a different ordering, it is unlikely that the reduced forms would come out identical. This leaves two possibilities:

[a] Reduce both bases under various orderings, take whichever results complete first, and convert them to bases under a common ordering, or

[b] Simultaneously reduce both bases under each ordering, and terminate when *both* have been reduced under the same ordering.

Consider [a]. It is not known whether or not it is "easy" to convert a reduced Gröbner-basis under one ordering into a reduced Gröbner-basis under another. The number of expressions can change, and the degree can jump from d to d^{n-1}, where n is the number of variables [8]. Table 5 shows the time it took to reduce directly under a "slow" order, vs. reducing under a "fast" order and re-reducing under the "slow" one.

Table 5 – Time (in Seconds) to Reduce Directly vs. Indirectly

	Case 7	Case 11
Direct Reduction		
Ordering:	$x > y$	$x > y > z$
Time:	11.350	173.234
Indirect Reduction		
First Ordering:	$y > x$	$z > x > y$
Second Ordering:	$x > y$	$x > y > z$
Time:	1.283	2668.434

Table 5 shows that the indirect route can be 9 times faster or 15 times slower. The potential speedup might make this collusive approach seem plausible, but the number of alternative combinations of first and second orderings grows too quickly.

Alternative [b] is viable under the conditions that seem to be present: the times to reduce a basis can vary dramatically with the ordering used, and converting a reduced basis to a different ordering can be as hard as the initial basis reduction. How well it works would depend on the nature of the input: whether some ordering is suitable for quickly reducing both.

8 Conclusions

We can draw the following conclusions at this point:

- "Obvious" ways of parallelizing Grõbner-basis reduction, such as generating S-polynomials or performing reductions in parallel sweeps are ineffective. Reducing under different orderings has some limited potential.
- Algorithmic "hacks" are effective in speeding up the process to a small degree; these give better performance than parallelism does, and seem to be themselves difficult to parallelize.

There is still room for exploring other ways to parallelize Grõbner-basis reduction; we have explored the obvious ways and they are not obviously effective.

Gonnet and Monagan [5] describe a general equation-solver which uses Grõbner-basis reduction (to solve the algebraic cases) in conjunction with other methods. Parallelism could be used to simultaneously try Grõbner-basis reduction in conjunction with resultant or other approaches. Parallelism may also be applied to other portions of the general equation-solver, such as searching for inconsistent subsets of a system of equations. These higher-level heuristic approaches seem more likely to provide payoffs in general problem solving.

9 Acknowledgments

Thanks to Manuel Bronstein for providing the benchmarks used in this paper. Both Eric Kaltofen and he lent some insight into the behavior of Buchberger's algorithm. Special thanks to Richard Fateman who directed this research, as well as guiding revisions to this paper.

This work was supported in part by the Army Research Office, grant DAAG29-85-K-0070, through the Center for Pure and Applied Mathematics, University of California, Berkeley, and the Defense Advanced Research Projects Agency (DoD) ARPA order #4871, monitored by Space & Naval Warfare Systems Command under contract N00039-84-C-0089, through the Electronics Research Laboratory, University of California, Berkeley.

References

[1] Zacharias, G. *Generalized Grõbner Bases in Commutative Polynomial Rings.* Bachelor thesis, Lab. for Computer Science, MIT, Cambridge, 1978.

[2] Buchberger, B. Grõbner Bases: An Algorithmic Method in Polynomial Ideal Theory. *Multidimensional Systems Theory,* N.K. Bose, Ed. D. Reidel Publishing Co., 1985, pp. 184-232.

[3] Bronstein, M. Gsolve: a Faster Algorithm for Solving Systems of Algebraic Equations. In *Proceedings of the 1986 ACM Symposium on Symbolic and Algebraic Computation (SYMSAC '86)* (Waterloo, Ontario, July 1986), B.W. Char, Ed. ACM, New York, 1986, pp. 247-249.

[4] Winkler, F., Buchberger, B., and Lichtenberger, F. Algorithm 628: An Algorithm for Constructing Canonical Bases of Polynomial Ideals. *ACM Transactions on Mathematical Software 11,* 1 (March 1985), 66-78.

[5] Gonnet, G.H. and Monagan, M.B. Solving Systems of Algebraic Equations, or the Interface between Software and Mathematics. In J. Davenport, Ed. *Proc. 1986 (AAAI/SIGSAM) Conf. on Computers and Mathematics,* Stanford (July 30 - Aug. 1).

[6] Huynh, D.T. A Superexponential Lower Bound for Gröbner Bases and Church-Rosser Commutative Thue Systems. *Journal of Information and Control,* 68:1-3, pp. 196-206 (1986).

[7] Watt, S.M. *Bounded Parallelism in Computer Algebra.* Rep. No. CS-86-12, Dept. of Mathematics, Univ. of Waterloo, Waterloo, May 1986.

[8] Moller, H.M. and Mora, F. Upper and Lower Bounds on the Degree of Gröbner Bases. In *Proceedings of Symbolic and Algebraic Computation (EUROSAM '84),* (Cambridge, July 1984). J.P. Fitch, Ed. *Lecture Notes in Computer Science,* G. Goos and J. Hartmanis, Ed. Springer-Verlag, New York, 1984. pp. 172-183.

[9] Czapor, S.R. and Geddes, K.O. On Implementing Buchberger's Algorithm for Gröbner Bases. In *Proceedings of the 1986 ACM Symposium on Symbolic and Algebraic Computation (SYMSAC '86)* (Waterloo, Ontario, July 1986), B.W. Char, Ed. ACM, New York, 1986, pp. 247-249.

10 Appendix – test cases 1-12

Case 1: $$\left\{ \begin{array}{l} x - ats \\ y^2 - a^2t^2(1-s^2) \\ z^2 - b^2(1-t^2) \end{array} \right\}$$ w.r.t. {x,y,z,t,s}

Case 2: $$\left\{ \begin{array}{l} 4x^2 + xy^2 - z + 1/4 \\ 2x + y^2z + 1/2 \\ -x^2z + x/2 + y^2 \end{array} \right\}$$ w.r.t. {x,y,z}

Case 3: $$\left\{ \begin{array}{l} z^2 - y^2/2 - x^2/2 \\ xz + xy - 2z \\ x^2 - y \end{array} \right\}$$ w.r.t. {x,y,z}

Case 4: $$\left\{ \begin{array}{l} x^4y^4 + y^6 - x^2y^4 - x^4y^2 + x^6 - y^4 + 2x^2y^2 - x^4 \\ x^3y^4 - xy^4/2 - x^3y^2 + 3x^5/2 + xy^2 - x^3 \\ x^4y^3 + 3/2y^5 - x^2y^3 - x^4y/2 - y^3 + x^2y \end{array} \right\}$$ w.r.t. {x,y}

Case 5: $$\left\{ \begin{array}{l} x^2 + y^2 + z^2 - t \\ y^2 - y + z - x^2 - x - 1 \\ x^2y^2 - 1 \end{array} \right\}$$ w.r.t. {x,y,z}

Case 6: $$\left\{ \begin{array}{l} x^2 + y^2 + z^2 - t \\ y^2 - y + z - x^2 - x - s \\ x^2y^2 - 1 \end{array} \right\}$$ w.r.t. {x,y,z}

Case 7: $$\left\{ \begin{array}{l} 2y^2(y^2 + x^2) + (b^2 - 3a^2)y^2 - 2by^2(y + x) + 2a^2b(y + x) - a^2x^2 + a^2(a^2 - b^2) \\ 4y^3 + 4y(y^2 + x^2) - 2by^2 - 4by(y + x) + 2(b^2 - 3a^2)y + 2a^2b \\ 4xy^2 - 2by^2 - 2a^2x + 2a^2b \end{array} \right\}$$ w.r.t. {x,y}

Case 8: $$\left\{ \begin{array}{l} ax^2 + bxy + cx + dy^2 + ey + f \\ bx^2 + 4dxy + 2ex + gy^2 + hy + k \end{array} \right\}$$ w.r.t. {x,y}

Case 9: $$\left\{ \begin{array}{l} 8x^2 - 2xy - 6xz + 3x + 3y^2 - 7yz + 10y + 10z^2 - 8z - 4 \\ 10x^2 - 2xy + 6xz - 6x + 9y^2 - yz - 4y - 2z^2 + 5z - 9 \\ 5x^2 + 8xy + 4xz + 8x + 9y^2 - 6yz + 2y - z^2 - 7z + 5 \end{array} \right\}$$ w.r.t. {x,y,z}

Case 10: $$\left\{ \begin{array}{l} x^2 + ayz + g \\ y^2 + bzx + h \\ z^2 + czy + k \end{array} \right\}$$ w.r.t. {x,y,z}

Case 11: $$\left\{ \begin{array}{l} x^2 + ayz + dx \\ y^2 + bzx + ay \\ z^2 + czy + fz \end{array} \right\}$$ w.r.t. {x,y,z}

Case 12: $$\left\{ \begin{array}{c} x^2 + yz + dx + 1 \\ y^2 + zx + ay + 1 \\ z^2 + xy + fz + 1 \end{array} \right\}$$ w.r.t. {x,y,z}